Azzolini Riccardo 2018-10-19

Espressioni e tipi numerici interi

1 Effetti collaterali

Un'espressione ha un **effetto collaterale** se la sua valutazione provoca un cambiamento dello stato dell'esecutore (cioè modifica il valore di una variabile, esegue input o output, ecc.).

Nella programmazione imperativa (e quindi anche in quella ad oggetti) gli effetti collaterali sono inevitabili, ma all'interno delle espressioni è necessario usarli con molta prudenza, dato che possono peggiorare la leggibilità del codice.

1.1 Esempi

```
int i, j;
i = 3;
j = i + (i = 5);
```

- `i` contiene 5
- `j` contiene 8

```
int i, j;
i = 3;
j = (i = 5) + i;
```

- `i` contiene 5
- `j` contiene 10

Esempio di utilizzo pratico:

```
int v;
while ((v = in.readInt()) != 0) { ... }
```

2 Abbreviazioni dell'operatore di assegnamento

Se `x` e `y` sono variabili numeriche, l'espressione

```
x = x + y
```

può essere scritta in forma più compatta utilizzando l'operatore `+=`:

```
x += y
```

Operatori analoghi sono disponibili anche per le altre operazioni aritmetiche (`-=`, `*=`, `/=`, `%=`).

3 Operatori di incremento e decremento

- Sono operatori *unari*.
- Hanno semantica diversa a seconda che vengano usati come *prefissi* o *postfissi*.
- Si applicano a variabili di tipo numerico.
- La valutazione di espressioni che li contengono da luogo a effetti collaterali.

3.1 Notazione prefissa

```
++i; --i;
```

Semantica operazionale:

1. Viene incrementata (`++i`) o decrementata (`--i`) la variabile.
2. Viene valutata l'espressione.

3.1.1 Esempio

```
i = 1;
j = ++i;
```

- `i` contiene 2
- `j` contiene 2

3.2 Notazione postfissa

```
i++; i--;
```

Semantica operazionale:

1. Viene valutata l'espressione.
2. Viene incrementata o decrementata la variabile.

3.2.1 Esempio

```
i = 1;
j = i++;
```

- `i` contiene 2
- `j` contiene 1

3.3 Esempi più complessi

```
int x = 3, y = 4, z;

x++ + y + x; // 3 + 4 + 4 = 11, x = 4

++x + y + x; // 4 + 4 + 4 = 12, x = 4

if (x++ == --y) // 3 == 3 -> true, x = 4, y = 3
    z = x + y; // z = 4 + 3 = 7
else
    z = x - y;
```

4 Precedenza e associatività

La **precedenza** degli operatori specifica il grado di priorità di un operatore rispetto a un altro, e di conseguenza l'ordine secondo il quale essi vengono applicati. Tale ordine può essere modificato usando le parentesi tonde.

Tra due operatori con la stessa precedenza, l'ordine di applicazione dipende dalle regole di **associatività**.

- Tutti gli operatori binari *tranne l'assegnamento* hanno associatività **left-to-right**: ad esempio, l'espressione `a + b + c` viene valutata come `(a + b) + c`.

- L'assegnamento e tutti gli operatori unari hanno associatività **right-to-left**: ad esempio, `a = b += c = -d` viene valutata come `a = (b += (c = (-d)))`.

5 Lazy evaluation

Normalmente, il risultato di un operatore è determinato dopo la valutazione di tutti gli operandi.

Ci sono però alcune eccezioni:

- gli operatori booleani `&&` e `||`
- l'*operatore condizionale*

Per questi operatori si applica un meccanismo chiamato **lazy evaluation** (valutazione "pigra" o *cortocircuitata*): se la valutazione del primo operando è sufficiente a determinare il risultato (ad esempio `false && x` o `true || x`), il secondo non viene mai valutato.

Esiste una versione degli operatori booleani che valuta sempre in modo completo le espressioni: `&` e `|`.

6 Operatore condizionale

`condizione ? espressione1 : espressione2`

- È un operatore ternario.
- La `condizione` è un'espressione booleana.
- `espressione1` e `espressione2` sono espressioni dello stesso tipo (o comunque di tipi compatibili)

L'espressione risultante ha

- tipo: il tipo di `espressione1` e `espressione2`
- valore:
  - il valore di `espressione1` se la `condizione` viene valutata `true`
  - il valore di `espressione2` se la `condizione` viene valutata `false`

7 Tipi numerici interi

- Rappresentano numeri interi con segno.
- Si distinguono i per range di valori rappresentabili, che dipendono dalla quantità di memoria occupata dalle variabili di ciascun tipo e sono *fissati dal linguaggio* (quindi sono indipendenti dall'architettura).

Tipo	Dimensione	Intervallo	Tot. valori
`byte`	8 bit	$[-2^7, 2^7 - 1]$	2^8
`short`	16 bit	$[-2^{15}, 2^{15} - 1]$	2^{16}
`int`	32 bit	$[-2^{31}, 2^{31} - 1]$	2^{32}
`long`	64 bit	$[-2^{63}, 2^{63} - 1]$	2^{64}

7.1 Letterali

- Possono essere preceduti dal segno `+` o `-`.
- Per il tipo `long` è necessario il suffisso `l` o `L`.
- Il compilatore verifica che il valore rappresentato da un letterale rientri nell'intervallo rappresentabile del particolare tipo.

7.2 Operatori binari

I principali operatori binari per i tipi numerici interi sono `+ - * / %`.

Il tipo delle espressioni risultanti dipende dal tipo degli operandi:

- se sono entrambi `byte` o `short`, l'espressione è di tipo `int`
- se sono entrambi `int`, l'espressione è di tipo `int`
- se sono entrambi `long`, l'espressione è di tipo `long`
- se sono di tipi diversi, quello di tipo più ristretto viene convertito al tipo più ampio prima di eseguire l'operazione